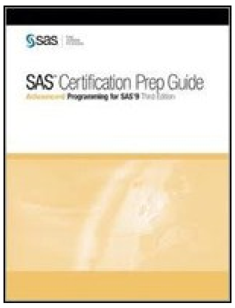


SAS Certification Prep Guide: Advanced Programming for SAS 9, Third Edition

by SAS Institute

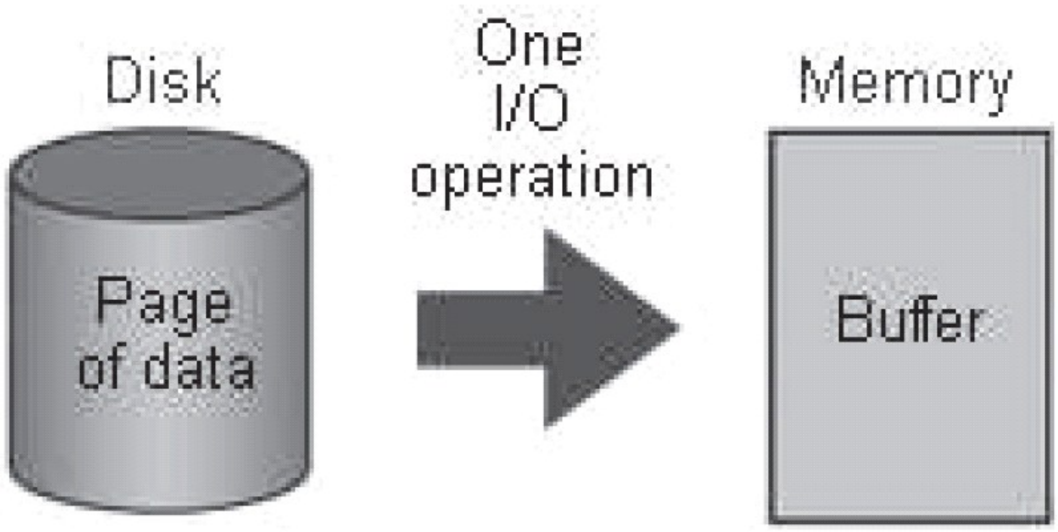
Chapter 20: Controlling Memory Usage

Overview

Introduction

As you have learned, there is no single set of programming techniques that is most efficient or appropriate in all situations. However, if reducing execution time is an important consideration in your computing environment, one way of achieving that goal is to reduce the number of times SAS has to read from or write to the storage medium.

In this chapter you learn to use options and a statement to control the size and number of data buffers, which in turn can affect your programs' execution times by reducing the number of I/O operations that SAS must perform.



Objectives

In this chapter, you learn to

- control the amount of data that is loaded into memory with each I/O transfer
- reduce I/O by holding a SAS data file in memory through multiple steps of a program.

Prerequisites

Before beginning this chapter, you should complete the following chapters:

- Part 1: SQL Processing with SAS
    - "Performing Queries Using PROC SQL" on page 4
    - "Performing Advanced Queries Using PROC SQL" on page 29
    - "Combining Tables Horizontally Using PROC SQL" on page 86
    - "Combining Tables Vertically Using PROC SQL" on page 132
    - "Creating and Managing Tables Using PROC SQL" on page 175
    - "Creating and Managing Indexes Using PROC SQL" on page 238
    - "Creating and Managing Views Using PROC SQL" on page 260
    - "Managing Processing Using PROC SQL" on page 278

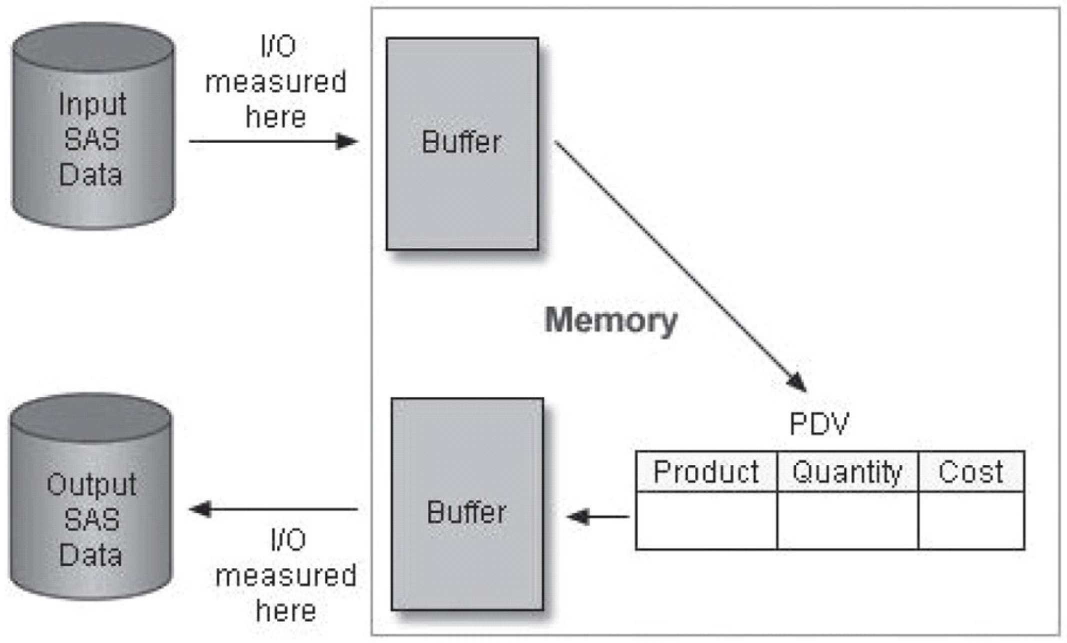
- Part 3: Advanced SAS Programming Techniques
  - "Creating Samples and Indexes" on page 470
  - "Combining Data Vertically" on page 502
  - "Combining Data Horizontally" on page 534
  - "Using Lookup Tables to Match Data" on page 580
  - "Formatting Data" on page 626
  - "Modifying SAS Data Sets and Tracking Changes" on page 656
- Part 4: Optimizing SAS Programs
  - "Introduction to Efficient SAS Programming" on page 701.

Controlling Page Size and the Number of Buffers

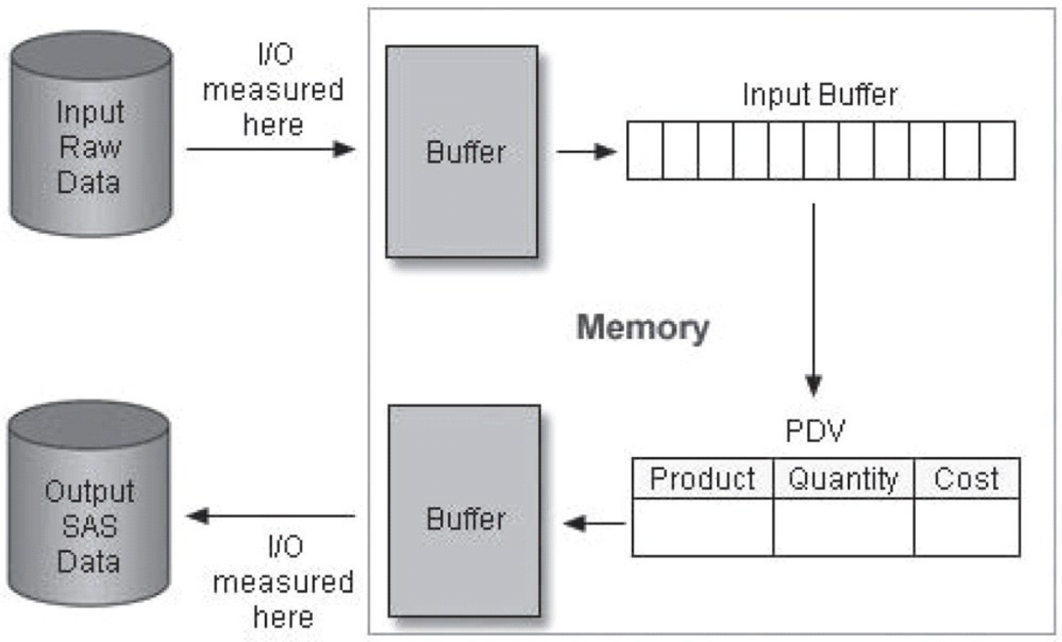
Measuring I/O

Improvement in I/O can come at the cost of increased memory consumption. In order to understand the relationship between I/O and memory, it is helpful to know when data is copied to a buffer and where I/O is measured. When you create a SAS data set using a DATA step,

1. SAS copies the data from the input data set to a buffer in memory
2. one observation at a time is loaded into the program data vector
3. each observation is written to an output buffer when processing is complete
4. the contents of the output buffer are written to the disk when the buffer is full.



The process for reading external files is similar. However, each record is first read into the input buffer before the data is parsed and read into the program data vector.



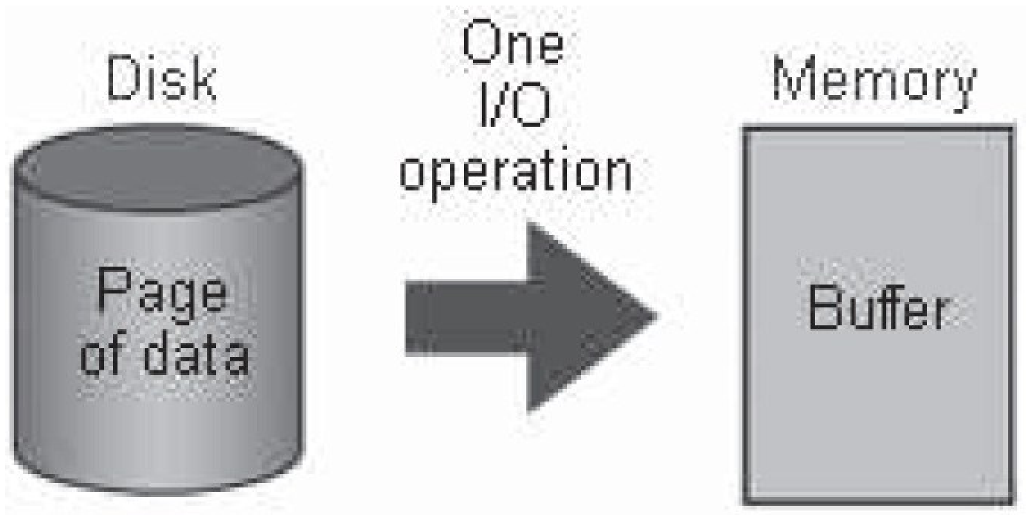
In both cases, I/O is measured when the input data is copied to the buffer in memory and when it is read from the output buffer to the output data set.

Page Size

Think of a buffer as a container in memory that is big enough for only one page of data. *A page*

- is the unit of data transfer between the storage device and memory
- is fixed in size when the data set is created, either to a default value or to a user-specified value.

The amount of data that can be transferred to one buffer in a single I/O operation is referred to as *page size*. Page size is analogous to buffer size for SAS data sets.



A larger page size can reduce execution time by reducing the number of times SAS has to read from or write to the storage medium. However, the improvement in execution time comes at the cost of increased memory consumption.

Reporting Page Size

You can use the CONTENTS procedure or the CONTENTS statement in the DATASETS procedure to report the page size and the number of pages.

Partial PROC CONTENTS Output

```
proc contents
     data=company.order_fact;
run;
```

Engine/Host Dependent Information	
Data Set Page Size	8192
Number of Data Set Pages	1231
First Data Page	1
Max Obs per Page	92
Obs in First Data Page	64
Number of Data Set Repairs	0
Filename	C:\company\order_fact.sas7bdat
Release Created	9.0101M3
Host Created	XP_PRO

The total number of bytes that a data file occupies equals the page size multiplied by the number of pages. For example, the page size for *Company.Order_fact* is *8192* and the number of pages is *9423*. Therefore, the data file occupies *77,193,216* bytes.

Note Note that the information that is available from PROC CONTENTS depends on the operating environment.

Note Page size is analogous to buffer size for SAS data sets.

Note In uncompressed data files, there is a 40-byte overhead (in a 64-bit operating environment) or a 24-byte overhead (in a 32-bit operating environment) at the beginning of each page and a 1-bit per observation overhead (rounded up to the nearest byte), used to denote an observation's status as deleted or not deleted, at the end of each page. You can learn about the structure of uncompressed and compressed data files in "Controlling Data Storage Space" on page 730.

Using the BUFSIZE= Option

To select a default page size, SAS uses an algorithm that is based on observation length, engine, and operating environment. The default page size is optimal for most SAS activities, especially on computers that are supporting multiple SAS jobs concurrently. However, in some cases, choosing a page/buffer size that is larger than the default can speed up execution time by reducing the number of times that SAS must read from or write to the storage medium.

You can use the *BUFSIZE=* system option or data set option to control the page size of an output SAS data set. BUFSIZE= specifies not only the page size (in bytes), but also the size of each buffer that is used for reading or writing the SAS data set. The new buffer size is a permanent attribute of the data set. After it is specified, it is used whenever the data set is processed.

General form, BUFSIZE= option:

BUFSIZE= MIN| MAX | *n;*

where

MIN

sets the page size to the smallest possible number in your operating environment.

MAX

sets the page size to the maximum possible number in your operating environment.

n

specifies the page size in bytes. For example, a value of 8 specifies a page size of 8 bytes, and a value of 4K specifies a page size of 4096 bytes. The default is 0, which causes SAS to use the optimal page size for the operating environment.

Caution MIN might cause unexpected results and should be avoided. Use BUFSIZE=0 to reset the buffer page size to the default value in your operating environment.

Note The syntax that is shown here applies to the OPTIONS statement. On the command line or in a configuration file, the syntax is specific to your operating environment. For details, see the SAS documentation for your operating environment.

Only certain page/buffer size values are valid for each operating environment. If you request an invalid value for your operating environment, SAS automatically rounds up to the next valid page/buffer size. BUFSIZE=0 is interpreted as a request for the default page/buffer size.

In the following program, the BUFSIZE= system option specifies a page size of *30720* bytes.

```
options bufsize=30720;
filename orders 'c:\orders.dat';
data company.orders_fact;
   infile orders;
   <more SAS code>
run;
```

Before you change the default page size, it is important to consider the access pattern for the data as well as the I/O transfer rate of the underlying hardware. In some cases, increasing the page size might degrade performance, particularly when the data is processed using direct (random) access.

Note The default value for BUFSIZE= is determined by your operating environment and is set to optimize sequential access. To improve performance for direct access, you should change the value for BUFSIZE=. For the default setting and possible settings for direct access, see the BUFSIZE= system option in the SAS documentation for your operating environment.

Note You can override the BUFSIZE= system option by using the BUFSIZE= data set option.

Caution If you use the COPY procedure to copy a data set to a library that is accessed via a different engine, the original page/buffer size is *not* necessarily retained.

Using the BUFNO= Option

You can use the BUFNO= system or data set option to control the number of buffers that are available for reading or writing a SAS data set. By increasing the number of buffers, you can control how many pages of data are loaded into memory with each I/O transfer.

Note Increasing the number of buffers might not affect performance under the Windows and UNIX operating environments, especially when you work with large data sets. By default, the Windows and UNIX operating environments read one buffer at a time. Under the SAS 9 Windows environment, you can override this default by turning on the SGIO system option when you invoke SAS. For details on the SGIO system option, see the SAS documentation for the Windows operating environment.

The following techniques might help to minimize I/O consumption:

- When you work with a small data set, allocate as many buffers as there are pages in the data set so that the entire data set can be loaded into memory. This technique is most effective if you read the same observations several times during processing.
- Under the z/OS operating environment, increase the number of buffers allocated, rather than the size of each buffer, as the size of the data set increases.

General form, BUFNO= option:

BUFNO= MIN| MAX |*n;*

where

MIN

sets the minimum number of buffers to 0, which causes SAS to use the minimum optimal value for the operating environment. This is the default.

MAX

sets the number of buffers to the maximum possible number in your operating environment, up to the largest four-byte, signed integer, which is 2^{31}-1, or approximately 2 billion.

n

specifies the number of buffers to be allocated.

Caution The recommended maximum for this option is 10.

Note The syntax that is shown here applies to the OPTIONS statement. On the command line or in a configuration file, the syntax is specific to your operating environment. For details, see the SAS documentation for your operating environment.

In the following program, the BUFNO= system option specifies that *4* buffers are available.

```
options bufno=4;
filename orders 'c:\orders.dat';
data company.orders_fact;
   infile orders;
   <more SAS code>
run;
proc print data=company.orders_fact;
run
```

The buffer number is not a permanent attribute of the data set and is valid only for the current step or SAS session.

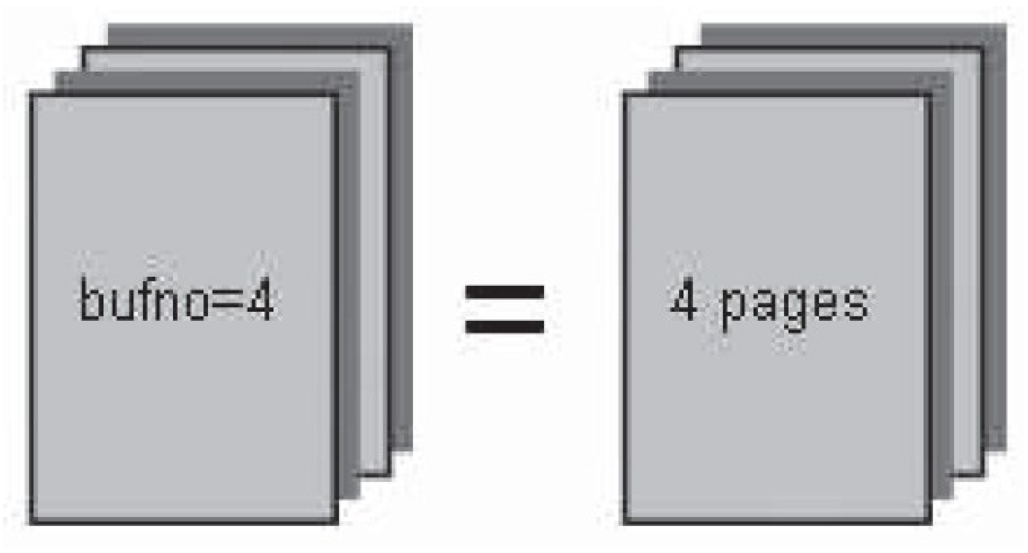


Figure 20.1: Current SAS Session

Note You can override the BUFNO= system option by using the BUFNO= data set option.

Note In SAS 9 and later, the BUFNO= option has no effect on thread-enabled procedures under the z/OS operating environment.

The product of BUFNO= and BUFSIZE=, rather than the specific value of either option, determines how much data can be transferred in one I/O operation. Increasing the value of either option increases the amount of data that can be transferred in one I/O operation.

BUFSIZE	BUFNO	Bytes Transferred in One I/O Operation
6144	2	12,288
6144	10	61,440
30,720	2	61,440
30,720	10	307,200

The number of buffers and the buffer size have a minimal effect on CPU usage.

Comparative Example: Using the BUFSIZE= Option and the BUFNO= Option

Settings for the Examples

Suppose you want to compare the resource usage when a data set is read using different buffer sizes and a varying number of buffers. The following sample programs use the following settings for the BUFNO= option and the BUFSIZE= option.

1. BUFSIZE=6144,BUFNO=2
2. BUFSIZE=6144, BUFNO=5
3. BUFSIZE=6144, BUFNO=1O
4. BUFSIZE=12288, BUFNO=2
5. BUFSIZE=12288, BUFNO=5
6. BUFSIZE=12288,BUFNO=10

You can use these samples as models for creating benchmark programs in your own environment. Your results might vary depending on the structure of your data, your operating environment, and the resources that are available at your site. You can also view general recommendations for controlling page size and the number of buffers.

Note 6144 bytes is the default page size under the z/OS operating environment.

Programming Techniques

---

❶ **OBUFSIZE=6144, BUFNO=2**

This program reads the data set *Retail.Order_fact* and creates the data set *Work.Orders*. The BUFSIZE= option specifies that *Work.Orders* is created with a buffer size of 6144 bytes. The BUFNO= option specifies that 2 pages of data are loaded into memory with each I/O transfer.

```
data work.orders (bufsize=6144 bufno=2);
   set retail.order_fact;
run;
```

❷ **BUFSIZE=6144, BUFNO=5**

This program reads the data set *Retail.Order_fact* and creates the data set *Work.Orders*. The BUFSIZE= option specifies that *Work.Orders* is created with a buffer size of 6144 bytes. The BUFNO= option specifies that *5* pages of data are loaded into memory with each I/O transfer.

```
data work.orders (bufsize=6144 bufno=5);
   set retail.order_fact;
run;
```

❸ **BUFSIZE=6144, BUFNO=10**

This program reads the data set *Retail.Order_fact* and creates the data set *Work.Orders*. The BUFSIZE= option specifies that *Work.Orders* is created with a buffer size of 6144 bytes. The BUFNO= option specifies that *10* pages of data are loaded into memory with each I/O transfer.

```
data work.orders (bufsize=6144 bufno=10);
   set retail.order_fact;
run;
```

❹ **OBUFSIZE=12288, BUFNO=2**

This program reads the data set *Retail.Order_fact* and creates the data set *Work.Orders*. The BUFSIZE= option specifies that *Work.Orders* is created with a buffer size of l2288 bytes. The BUFNO= option specifies that *2* pages of data are loaded into memory with each I/O transfer.

```
data work.orders (bufsize=12288 bufno=2);
   set retail.order_fact;
run;
```

❺ **BUFSIZE=12288, BUFNO=5**

This program reads the data set *Retail.Order_fact* and creates the data set *Work.Orders*. The BUFSIZE= option specifies that *Work.Orders* is created with a buffer size of l2288 bytes. The BUFNO= option specifies that *5* pages of data are loaded into memory with each I/O transfer.

```
data work.orders (bufsize=12288 bufno=5);
   set retail.order_fact;
run;
```

❻ **BUFSIZE=12288, BUFNO=10**

This program reads the data set *Retail.Order_fact* and creates the data set *Work.Orders*. The BUFSIZE= option specifies that *Work.Orders* is created with a buffer size of l2288 bytes. The BUFNO= option specifies that *10* pages of data are loaded into memory with each I/O transfer.

```
data work.orders (bufsize=12288 bufno=10);
   set retail.order_fact;
run;
```

General Recommendations

- To reduce I/O operations on a small data set, allocate as many buffers as there are pages in the data set so that the entire data set can be loaded into memory. This technique is most effective if you read the same observations several times during processing.
- Under the z/OS operating environment, as the size of the data set increases, increase the number of buffers allocated, rather than the size of each buffer, to minimize I/O consumption.

Using the SASFILE Statement

Overview

Another way of improving performance is to use the SASFILE statement to hold a SAS data file in memory so that the data is available to multiple program steps. Keeping the data file open reduces open/close operations, including the allocation and freeing of memory for buffers.

General form, SASFILE statement:

SASFILE *SAS-data-file <(password-option(s))>* **OPEN | LOAD | CLOSE;**

where

SAS-data-file

is a valid SAS data file (a SAS data set with the member type DATA).

password-option(s)

specifies one or more password options.

OPEN

opens the file and allocates the buffers, but defers reading the data into memory until a procedure or statement is executed.

LOAD

opens the file, allocates the buffers, and reads the data into memory.

CLOSE

closes the file and frees the buffers.

The SASFILE statement opens a SAS data file and allocates enough buffers to hold the entire file in memory. Once the data file is read, the data is held in memory, and it is available to subsequent DATA and PROC steps or applications until either

- a SASFILE CLOSE statement frees the buffers and closes the file
- the SAS session ends, which automatically frees the buffers and closes the file.

In the following program, the SASFILE statement opens the SAS data file *Company.Sales*, allocates the buffers, and reads the data into memory.

```
sasfile company.sales load;
proc print data=company.sales;
   var Customer_Age_Group;
run;
proc tabulate data=company.sales;
   class Customer_Age_Group;
   var Customer_BirthDate;
   table Customer_Age_Group,Customer_BirthDate*(mean median);
run;
sasfile company.sales close;
```

Note The SASFILE statement can also be used to reduce CPU time and I/O in SAS programs that repeatedly read one or more SAS data views. Use a DATA step to create a SAS data file in the *Work* library that contains the view's result set. Then use the SASFILE statement to load that data file into memory.

Note Though a file that is opened with the SASFILE statement can be used for subsequent input or update processing, it cannot be used for subsequent utility or output processing. For example, you cannot replace the file or rename its variables.

Guidelines for Using the SASFILE Statement

When the SASFILE statement executes, SAS allocates the number of buffers based on the number of pages for the data file and index file. If the file in memory increases in size during processing because of changes or additions to the data, the number of buffers also increases.

It is important to note that I/O processing is reduced only if there is sufficient real memory. If there is not sufficient real memory, the operating environment might

- use virtual memory
- use the default number of buffers.

If SAS uses virtual memory, there might be a degradation in performance.

If you need to repeatedly process part of a SAS data tile and the entire file will not fit into memory, use a DATA step with the SASFILE statement to create a subset of the file that does fit into memory, and then process that subset repeatedly. This saves CPU time in the processing steps because those steps will read a smaller file, in addition to the benefit of the file being resident in memory.

Note When using a SASFILE statement, monitor the paging activity (the I/O activity that is done by the virtual memory management subsystem of your operating environment) while your program runs. If the paging activity increases substantially, consider keeping less data in memory and using techniques described elsewhere in this course to reduce memory requirements.

Comparative Example: Using the SASFILE Statement

Using Different Data File Sizes

Suppose you want to create multiple reports from SAS data files that vary in size. Using small, medium, and large data files, you can compare the resource usage when the PRINT, TABULATE, MEANS, and FREQ procedures are used with and without the SASFILE statement to create reports.

Name of Data File	Number of Rows	Page Size	Number of Pages	Number of Byes
Retail.Small	45,876	24, 576	540	13,279,232
Retail.Medium	458,765	24, 576	5, 398	132,669,440
Retail.Large	4,587,654	24, 576	53, 973	1,326,448,640

1. Small Data File without the SASFILE Statement
2. Medium Data File without the SASFILE Statement
3. Large Data File without the SASFILE Statement
4. Small Data File with the SASFILE Statement
5. Medium Data File with the SASFILE Statement
6. Large Data File with the SASFILE Statement.

The following sample programs show each of these techniques. You can use these samples as models for creating benchmark programs in your own environment. Your results might vary depending on the structure of your data, your operating environment, and the resources that are available at your site. You can also view general recommendations for using the SASFILE statement.

Programming Techniques

---

❶ Small Data File without the SASFILE Statement

This program creates reports using the PRINT, TABULATE, MEANS, and FREQ procedures. The SAS data file *Retail.Small* is opened and closed with each procedure.

```
proc print data=retail.small;
   where cs=100;
   var Customer_Age_Group;
run;
proc tabulate data=retail.small;
   class Customer_Age_Group;
   var Customer_BirthDate;
   table Customer_Age_Group,Customer_BirthDate*(mean median);
run;
proc means data=retail.small;
   var Customer_Age;
   class Customer_Group;
   output out=summary sum=;
run;
proc freq data=retail.small;
   tables Customer_Country;
run;
```

❷ Medium Data File without the SASFILE Statement

This program creates reports using the PRINT, TABULATE, MEANS, and FREQ procedures. The SAS data file

Retail.Medium is opened and closed with each procedure.

```
proc print data=retail.medium;
   where cm=100;
   var Customer_Age_Group;
run;
proc tabulate data=retail.medium;
   class Customer_Age_Group;
   var Customer_BirthDate;
   table Customer_Age_Group,Customer_BirthDate*(mean median);
run;
proc means data=retail.medium;
   var Customer_Age;
   class Customer_Group;
   output out=summary sum=;
run;
proc freq data=retail.medium;
   tables Customer_Country;
run;
```

❸ Large Data File without the SASFILE Statement

This program creates reports using the PRINT, TABULATE, MEANS, and FREQ procedures. The SAS data file *Retail.Large* is opened and closed with each procedure.

```
proc print data=retail.large;
   where cl=100;
   var Customer_Age_Group;
run;
proc tabulate data=retail.large;
   class Customer_Age_Group;
   var Customer_BirthDate;
   table Customer_Age_Group,Customer_BirthDate*(mean median);
run;
proc means data=retail.large;
   var Customer_Age;
   class Customer_Group;
   output out=summary sum=;
run;
proc freq data=retail.large;
   tables Customer_Country;
run;
```

❹ Small Data File with the SASFILE Statement

In this program, the SASFILE LOAD statement opens the SAS data file *Retail.Small* and loads the entire file into memory. The data is then available to the PRINT, TABULATE, MEANS, and FREQ procedures. The SASFILE CLOSE statement closes *Retail.Small* and frees the buffers.

```
sasfile retail.small load;
proc print data=retail.small;
   where cs=100;
   var Customer_Age_Group;
run;
proc tabulate data=retail.small;
   class Customer_Age_Group;
   var Customer_BirthDate;
   table Customer_Age_Group,Customer_BirthDate*(mean median);
run;
proc means data=retail.small;
   var Customer_Age;
   class Customer_Group;
   output out=summary sum=;
run;
proc freq data=retail.small;
   tables Customer_Country;
run;
sasfile retail.small close;
```

5 Medium Data File with the SASFILE Statement

In this program, the SASFILE LOAD statement opens the SAS data file *Retail.Medium* and loads the entire file into memory. The data is then available to the PRINT, TABULATE, MEANS, and FREQ procedures. The SASFILE CLOSE statement closes *Retail.Medium* and frees the buffers.

```
sasfile retail.medium load;
proc print data=retail.medium;
   where cm=100;
   var Customer_Age_Group;
run;
proc tabulate data=retail.medium;
   class Customer_Age_Group;
   var Customer_BirthDate;
   table Customer_Age_Group,Customer_BirthDate*(mean median);
run;
proc means data=retail.medium;
   var Customer_Age;
   class Customer_Group;
   output out=summary sum=;
run;
proc freq data=retail.medium;
   tables Customer_Country;
run;
sasfile retail.medium close;
```

6 Large Data File with the SASFILE Statement

In this program, the SASFILE LOAD statement opens the SAS data file *Retail.Large* and loads the entire file into memory. The data is then available to the PRINT, TABULATE, MEANS, and FREQ procedures. The SASFILE CLOSE statement closes *Retail.Large* and frees the buffers.

```
sasfile retail.large load;
proc print data=retail.large;
   where cl=100;
   var Customer_Age_Group;
run;
proc tabulate data=retail.large;
   class Customer_Age_Group;
   var Customer_BirthDate;
   table Customer_Age_Group,Customer_BirthDate*(mean median);
run;
proc means data=retail.large;
   var Customer_Age;
   class Customer_Group;
   output out=summary sum=;
run;
proc freq data=retail.large;
   tables Customer_Country;
run;
sasfile retail.large close;
```

General Recommendations

- If you need to repeatedly process a SAS data file that will fit entirely in memory, use the SASFILE statement to reduce I/O and some CPU usage.
- If you use the SASFILE statement and the SAS data file will not fit entirely in memory, the code will execute, but there might be a degradation in performance.
- If you need to repeatedly process part of a SAS data tile and the entire file will not fit into memory, use a DATA step with the SASFILE statement to create a subset of the file that does fit into memory, and then process that subset repeatedly. This saves CPU time in the processing steps because those steps will read a smaller file, in addition to the benefit of the file being resident in memory.

Additional Features

Using the IBUFSIZE= System Option

Beginning with SAS 9, you can use the IBUFSIZE= system option to specify the page size for an index file. Typically, you do not need to specify an index page size. However, you might need to use the IBUFSIZE= option if

- there are many levels in the index
- the length of an index value is very large.

The main resource that is saved when reducing levels in the index is I/O. If your application is experiencing a lot of I/O in the index file, increasing the page size might help. However, you must re-create the index file after increasing the page size. The number of pages that are required for the index varies with the page size, the length of the index value, and the values themselves.

General form, IBUFSIZE= system option:

IBUFSIZE= MAX | *n;*

where

MAX

sets the page size for an index file to the maximum possible number. For IBUFSIZE=, the maximum value is 32,767 bytes.

n

specifies the page size in bytes.

Note The MIN setting should be avoided.

When an index is used to process a request, such as for WHERE processing, SAS searches the index file in order to rapidly locate the requested record(s). The page size affects the number of levels in the index. The more pages there are, the more levels in the index. The more levels, the longer the index search takes. Increasing the page size allows more index values to be stored on each page, thus reducing the number of pages (and the number of levels).

Use IBUFSIZE=0 to reset the index page size to the default value in your operating environment.

Note For details on using the IBUFSIZE= system option, see the SAS documentation.

Summary

Controlling Page Size and the Number of Buffers

When you read a SAS data set or an external file, I/O is measured when the input data is copied to the buffer in memory and when it is read from the output buffer to the output data set.

A page is the unit of data transfer between the storage device and memory. When you create a SAS data set, SAS takes the data and copies it to a buffer. Each buffer can hold one page of data.

The amount of data that can be transferred to one buffer in a single I/O operation is referred to as the page size. Increasing the page size can speed up execution time by reducing the number of times SAS has to read from or write to the storage medium. You can use the CONTENTS procedure to report the page size and the number of pages.

You can use the BUFSIZE= system option or data set option to control the page size of an output SAS data set. The new buffer size is permanent. After it is specified, it is used whenever the data set is processed.

You can use the BUFNO= system or data set option to control how many buffers are available for reading or writing a SAS data set. By increasing the number of buffers, you can control how many pages of data are loaded into memory with each I/O transfer.

The product of BUFNO= and BUFSIZE=, rather than the specific value of either option, determines how much data can be transferred in one I/O operation. Increasing either option increases the amount of data that can be transferred in one I/O operation. However, the improvement in I/O comes at the cost of increased memory consumption.

Review the related comparative example:

- "Comparative Example: Using the BUFSIZE= Option and the BUFNO= Option" on page 718.

Using the SASFILE Statement

Another way of improving performance is to use the SASFILE statement to hold a SAS data file in memory so that the data is available to multiple program steps. Keeping the data set open reduces open/close operations, including the allocation and freeing of memory for buffers.

When the SASFILE statement executes, SAS allocates the number of buffers based on the number of pages for the data file and index file. If the file in memory increases in size during processing because of changes or additions to the data, the number of buffers also increases.

It is important to note that I/O processing is reduced only if there is sufficient real memory. If SAS uses virtual memory, there can be a degradation in performance.

Review the related comparative example:

- "Comparative Example: Using the SASFILE Statement" on page 721.

Additional Features

The IBUFSIZE= system option specifies the page size for an index file. Typically, you do not need to specify an index page size. However, you might need to use the IBUFSIZE= option if

- there are many levels in the index
- the length of an index value is very large.

The main resource that is saved when reducing levels in the index is I/O. If your application is experiencing a lot of I/O in the index file, increasing the page size might help. However, you must re-create the index file after increasing the page size. The number of pages that are required for the index varies with the page size, the length of the index value, and the values themselves.

Quiz

Select the best answer for each question. After completing the quiz, check your answers using the answer key in the appendix.

1. Which of the following statements is true regarding the BUFNO= option? ?

a. The BUFNO= option specifies the size of each buffer that is used for reading or writing a SAS data set.

b. The BUFNO= option can improve execution time by limiting the number of input/output operations that are required.

c. Using the BUFNO= option results in permanent changes to the data set.

d. Using the BUFNO= option to increase the number of buffers results in decreased memory consumption.

2. Which of the following statements is not true regarding a page? ?

a. A page is the unit of data transfer between the storage device and memory.

b. A page includes the number of bytes that are used by the descriptor portion, the data values, and the overhead.

c. The size of a page is analogous to buffer size.

d. The size of a page can be changed at any time.

3. The total number of bytes occupied by a data set equals…? ?

   a. the page size multiplied by the number of pages.
   b. the page size multiplied by the number of observations.
   c. the sum of the page size and the number of pages.
   d. the number of pages multiplied by the number of variables.

4. Which statement opens the file *Work.Quarter1*, allocates enough buffers to hold the entire file in memory, and reads the data into memory? ?
   a. `sasfile work.quarter1 open;`
   b. `sasfile work.quarter1 load;`
   c. `sasfile work.quarter1 bufno=max;`
   d. `sasfile work.quarter1 bufsize=max;`

5. Which of the following statements is true regarding a file that is opened with the SASFILE statement? ?
   a. The file is available to subsequent DATA and PROC steps or applications until a SASFILE CLOSE statement is executed or until the SAS session ends.
   b. The file is available to subsequent DATA and PROC steps or applications until a SASFILE END statement is executed.
   c. The file is available for subsequent utility or output processing until the program ends.
   d. If the file increases in size during processing, the number of buffers remains the same.

Answers

1. Correct answer: b

   You can use the BUFNO= system option or data set option to control how many buffers are available for reading or writing a SAS data set. Using BUFNO= can improve execution time by limiting the number of input/output operations that are required for a particular SAS data set. However, the improvement in I/O comes at the cost of increased memory consumption. The buffer number is not a permanent attribute of the data set and is valid only for the current step or SAS session.

2. Correct answer: a

   A page is fixed in size when the data set is created, either to a default value or a specified value. You can use the BUFSIZE= option to control the page size of an output SAS data set. BUFSIZE= specifies not only the page size (in bytes), but also the size of each buffer that is used for reading or writing the SAS data set. The new buffer size is permanent; after it is specified, it is used whenever the data set is processed.

3. Correct answer: a

   The total number of bytes occupied by a data set equals the page size multiplied by the number of pages. You can use the CONTENTS procedure to report the page size and the number of pages.

4. Correct answer: b

   The SASFILE LOAD statement opens the file, allocates the buffers, and reads the data into memory.

5. Correct answer: a

   When a SAS data file is opened using the SASFILE statement, the data is held in memory, and is available to subsequent DATA and PROC steps or applications, until either a SASFILE CLOSE statement is executed or the SAS session ends. Though a file that is opened with the SASFILE statement can be used for subsequent input or update processing, it cannot be used for subsequent utility or output processing. If the file in memory increases in size during processing, the number of buffers also increases.